

ChatGPT

OpenAI - huge systems, Largest language models

Lots of data

Model are very heavy

They take lot of time and infrastructure(computers, ram, hardspace)

ML: linear regression: House price prediction: $y = a_1 \cdot x_1 + \dots + a_n \cdot x_n$, x_1, \dots, x_n

House data

Process

S Clean

Transform

Store

Models

While modelling we track, hyperparameters, cross validation, overfitting, underfitting, track performance metrics like accuracy

Choose a final model

Test the model some time

We deploy the model

1. How do you store input data? Distributed systems, BigData
2. How do they clean it?
Find resources, find codes, datasets.

BIG DATA

Big data refers to more diverse data that arrives in higher amounts and moves faster. Big data refers to larger, more complicated data sets, particularly those derived from new sources. Because these data sets are so large. Although businesses have been gathering massive amounts of data for decades, the term "Big Data" only became popular in the early to mid-2000s.

Big data refers to data collections that are too massive or complicated for typical data-processing application software to handle. Data with more fields have more statistical power; however, data with more fields also have a higher false discovery rate.




TYPES OF DATA:

1. Structured - data fields with data type- ex: mysql data bases
2. Semi structured - data fields without datatype - ex csv files
3. Unstructured - doesn't contain columns - multi media

Big Data is data of very big size which can not be processed with usual tools.

Generally, we classify the problems related to the handling of data into three buckets:

1. Volume
2. Velocity
3. variety

VOLUME	VELOCITY	VARIETY
Data At Rest	Data In Motion	Data in Many Forms
		
Problems related to storage of huge data reliably. e.g. Storage of Logs of a website, Storage of data by gmail. FB: 300 PB. 600TB/ day	Problems Involving the handling of data coming at fast rate. e.g. Number of requests being received by Facebook, Youtube streaming, Google Analytics	Problems involving complex data structures e.g. Maps, Social Graphs, Recommendations

Data could be termed as Big Data if either Volume, Velocity or Variety becomes impossible to handle using traditional tools.

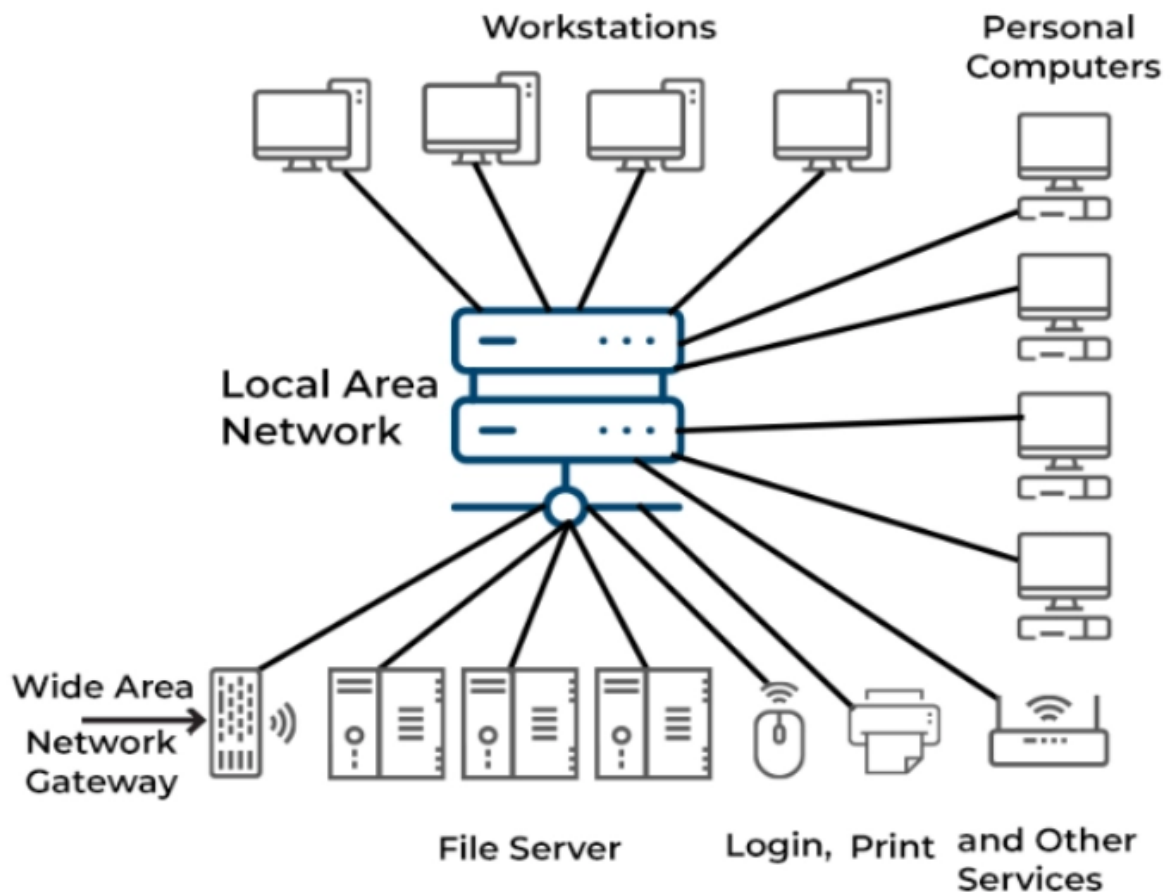
Speed of computation could be depend on 4 components:

- 1.cpu
- 2.ram
- 3.hard disk
- 4.network

While processing Big Data at least one of these four components becomes the bottleneck. For that purpose only we will use distributed systems

DISTRIBUTED SYSTEMS

In a distributed system, the components are spread across multiple networked computers and communicate and coordinate their actions by transferring messages from one system to the next.



BIG DATA SOLUTIONS:

1. The first and most powerful stack is Apache Hadoop and Spark together.
2. The second way could be to use Cassandra or MongoDB.
3. The third could be to use Google Compute Engine or Microsoft Azure.

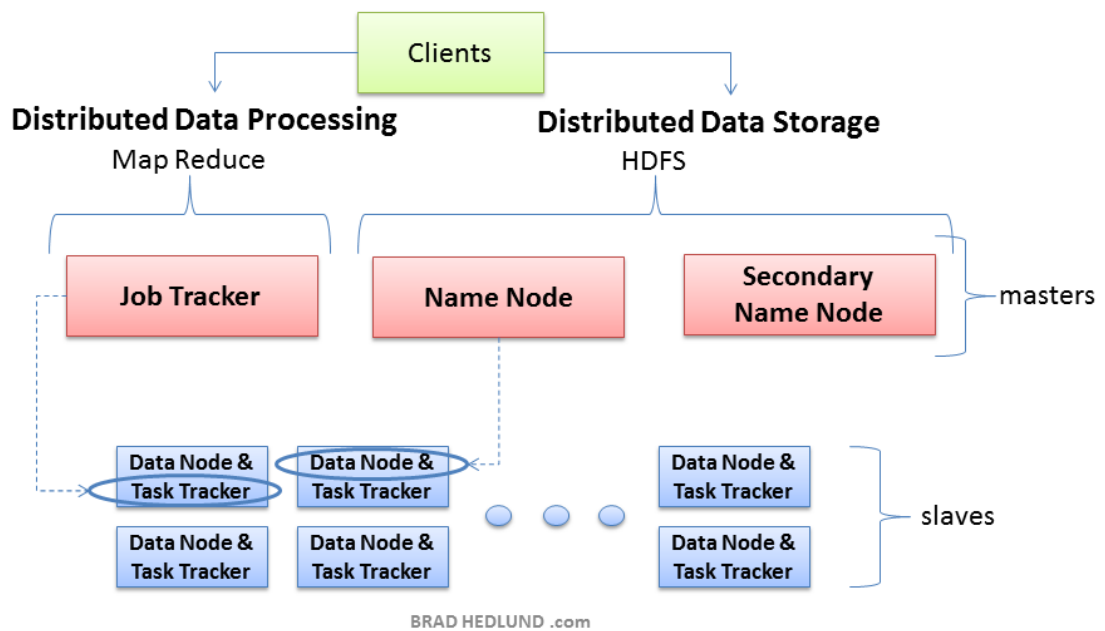
In such cases, you would have to upload your data to Google or Microsoft which may not be acceptable to your organization sometimes.

HADOOP:

Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering

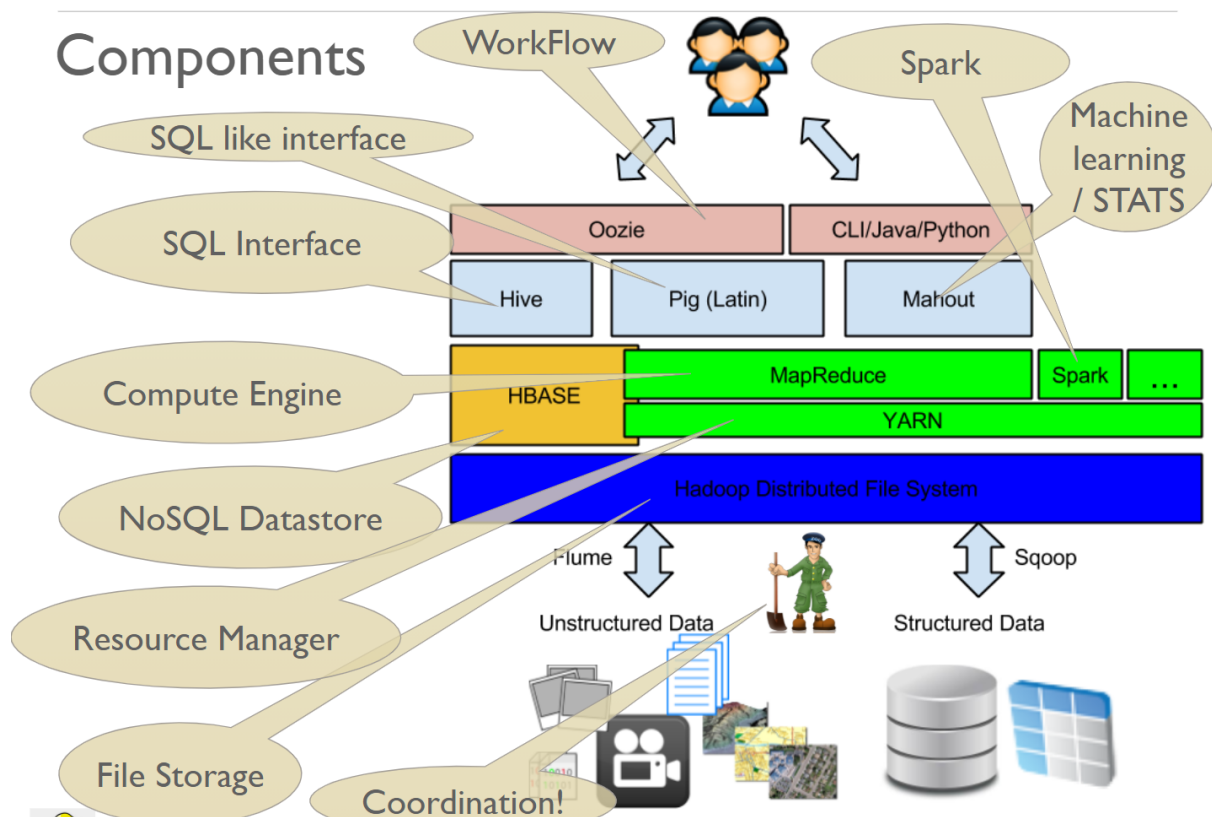
multiple computers to analyse massive datasets in parallel more quickly.

Hadoop Server Roles



It is a framework to handle Big Data. Hadoop is written in Java so that it can run on all kinds of devices.

- > distributed
- > scalable
- > reliable



HDFS: HDFS or Hadoop Distributed File System is the most important component because the entire ecosystem depends upon it. It is based on the Google File System.

YARN: keeps track of all the resources (CPU, Memory) of machines

HBase: provides database table. HBase is a NoSQL Datastore.

MapReduce: You write your programs in two parts: Map and reduce. The map part transforms the raw data into key-value and reduces part groups and combines data based on the key.

Spark: Spark is another computational framework similar to MapReduce but faster and more recent. Spark has its own huge stack.

Hive

Apache Hive makes it possible to write your logic in SQL which internally converts it into MapReduce.

Pig (Latin): Pig Latin is a simplified SQL like language to express your ETL needs in stepwise fashion. Pig is the engine that translates Pig Latin into Map Reduce and executes it on Hadoop.

Mahout: It is a library of machine learning algorithms that run in a distributed fashion.

ZooKeeper: Apache Zookeeper is an independent component which is used by various distributed frameworks such as HDFS, HBase, Kafka, YARN.

Flume: It loads unstructured data from many sources to a central source such as HDFS.

SQOOP: Sqoop is used to transport data between Hadoop and SQL Databases. Sqoop utilises MapReduce to efficiently transport data using many machines in a network.

Oozie: Since a project might involve many components, there is a need of a workflow engine to execute work in sequence.

SPARK:

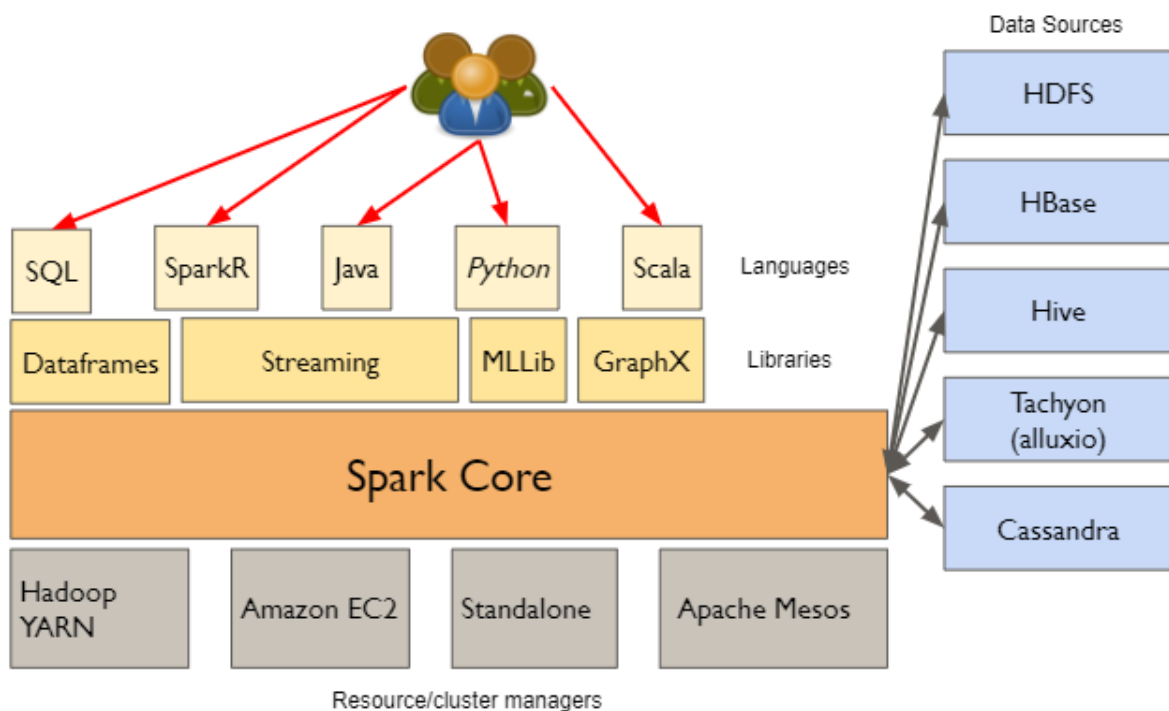
Apache Spark is a fast and general engine for large-scale data processing.

It is around 100 times faster than MapReduce using only RAM and 10 times faster if using the disk.

Advantages over hadoop:

- 1.In-memory Processing
- 2.Stream Processing
- 3.Less Latency
- 4.Lazy Evaluation
- 5.Less Lines of Code

Spark Architecture



Sources:

Instead of building its own file or data storages, Apache spark made it possible to read from all kinds of data sources: Hadoop Distributed File System, HBase, Hive, Tachyon, Cassandra.

Libraries:

Apache Spark comes with a great set of libraries.

Data frames provide a generic way to represent the data in the tabular structure. The data frames make it possible to query data using R or SQL instead of writing tons of code.

Streaming Library makes it possible to process fast incoming streaming of huge data using Spark.

MLLib is a very rich machine learning library. It provides very sophisticated algorithms which run in distributed fashion.

GraphX makes it very simple to represent huge data as a graph. It provides library of algorithms to process graphs using multiple computers.

Spark and its libraries can be used with **Scala, Java, Python, R, and SQL**.

HADOOP CODE:

1.Import the necessary libraries:

```
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.FileSystem;  
import org.apache.hadoop.fs.Path;
```

2. Set up the Hadoop configuration:

```
Configuration conf = new Configuration();  
conf.set("fs.defaultFS", "hdfs://localhost:9000");
```

3. Create a file system object:

```
FileSystem fs = FileSystem.get(conf);
```

4. Create a path object for the file you want to store:

```
Path inputPath = new Path("/path/to/input/file.txt");
```


5. Create a path object for the location where you want to store the file:

```
Path outputPath = new Path("/path/to/output/");
```

6. Check if the output path already exists:

```
if (fs.exists(outputPath)) {  
    System.out.println("Output path already exists.");  
    return;  
}
```

7. Create the output directory:

```
fs.mkdirs(outputPath);
```

8. Copy the input file to the output directory:

```
fs.copyFromLocalFile(inputPath, outputPath);
```

9. Close the file system object:

```
fs.close()
```

CASSANDRA DISTRIBUTED SYSTEM:

cassandra is a distributed database system that is designed to handle large volumes of data across multiple nodes in a cluster. It is a highly scalable and fault-tolerant system that is used by many large-scale organisations.

Cassandra uses a distributed hash table (DHT) algorithm to partition the data across the nodes in the cluster.

CASSANDRA CODE:

```
// Import the necessary Cassandra libraries
```

```
import com.datastax.driver.core.Cluster;
```

```
import com.datastax.driver.core.Session;

// Set up the Cassandra cluster

String node = "localhost";

int port = 9042;

Cluster cluster = Cluster.builder().addContactPoint(node).withPort(port).build();

// Connect to the Cassandra cluster

Session session = cluster.connect();

// Create the keyspace

String keyspace = "my_keyspace";

String replicationStrategy = "SimpleStrategy";

int replicationFactor = 1;

session.execute("CREATE KEYSPACE IF NOT EXISTS " + keyspace +

    " WITH replication = {'class':" + replicationStrategy + ", 'replication_factor':" +

    replicationFactor + "}");

// Create the table

String table = "my_table";

session.execute("CREATE TABLE IF NOT EXISTS " + keyspace + "." + table + " (id

UUID PRIMARY KEY, name text, age int);");
```

```
// Insert data into the table
```

```
String id = UUID.randomUUID().toString();
```

```
String name = "John";
```

```
int age = 30;
```

```
session.execute("INSERT INTO " + keyspace + "." + table + " (id, name, age) VALUES (" + id + ", " + name + ", " + age + ");");
```

```
// Query the data from the table
```

```
ResultSet results = session.execute("SELECT * FROM " + keyspace + "." + table + ";");
```

```
for (Row row : results) {
```

```
    System.out.println("ID: " + row.getUUID("id") + ", Name: " + row.getString("name") + ", Age: " + row.getInt("age"));
```

```
}
```

```
// Close the session and cluster
```

```
session.close();
```

```
cluster.close();
```

Processing steps involves:

Step1: load data into cascandra using apache spark,kafka

Step2: create data model means defining tables, columns and keyspace

Step3: clean and preprocess the data using cql statements -aggregation, joins or filtering

Step4: analyse and visualise the data: using jupyter notebook or custom dashboards like power bi

code for Storing data using cloud storage :

```
// Import the necessary AWS SDK libraries

import com.amazonaws.auth.AWSSStaticCredentialsProvider;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.client.builder.AwsClientBuilder;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.ObjectMetadata;

import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;


// Set up your AWS credentials

String accessKey = "your_access_key_here";

String secretKey = "your_secret_key_here";

String region = "your_region_here";

String bucketName = "your_bucket_name_here";


// Set up the Amazon S3 client

BasicAWSCredentials credentials = new BasicAWSCredentials(accessKey,
secretKey);

AmazonS3 s3client = AmazonS3ClientBuilder.standard()
```

```
.withCredentials(new AWSStaticCredentialsProvider(credentials))

.withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(region,
""))

.build();

// Create a file object for the file you want to store

File file = new File("/path/to/your/file.txt");

// Create a PutObjectRequest object with the bucket name and key

PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
file.getName(), file);

// Set the content type of the file

ObjectMetadata metadata = new ObjectMetadata();

metadata.setContentType("text/plain");

putObjectRequest.setMetadata(metadata);

// Upload the file to the Amazon S3 bucket

s3client.putObject(putObjectRequest);

// Close the Amazon S3 client

s3client.shutdown();
```

DATA CLEANING :

Cleaning data in distributed systems can be a complex task, and the specific approach you take will depend on the tools and frameworks you're using, as well as the nature of the data you're working with.

1. Apache Hadoop - MapReduce
2. Apache Spark - operations on RDD
3. Apache Flink - Flink filtering operators
4. Apache Storm - Storm filtering operators
5. Google Cloud Dataflow - Dataflow pipelines

However, here's an example of a basic data cleaning pipeline using Apache Spark.

CODE : (Apache Spark)

```
// Import the necessary Spark libraries

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;


// Set up a Spark configuration object

SparkConf conf = new SparkConf()

    .setAppName("DataCleaning")

    .setMaster("local");


// Set up a Spark context object

JavaSparkContext sc = new JavaSparkContext(conf);
```

```
// Load your data into an RDD

JavaRDD<String> inputRDD = sc.textFile("hdfs://path/to/input/data");


// Remove any blank lines from the RDD

JavaRDD<String> cleanedRDD = inputRDD.filter(line -> !line.trim().isEmpty());


// Split each line into fields using a delimiter

JavaRDD<String[]> splitRDD = cleanedRDD.map(line -> line.split(","));


// Remove any lines with missing or invalid data

JavaRDD<String[]> filteredRDD = splitRDD.filter(fields -> fields.length == 3 &&
isValid(fields[2]));


// Define a function to check if a field is valid

private static boolean isValid(String field) {

    return field != null && !field.trim().isEmpty() && field.matches("^([a-zA-Z]+)$");

}


// Save the cleaned data to a new location

filteredRDD.saveAsTextFile("hdfs://path/to/cleaned/data");


// Shut down the Spark context

sc.close();
```

data cleaning using Apache Hadoop:

Step 1: Import the necessary Hadoop libraries and packages

```
import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

Step 2 : Define the MapReduce job with a Mapper and Reducer.

```
public class DataCleaning {

    public static class DataCleaningMapper

        extends Mapper<Object, Text, Text, Text>{

        private Text word = new Text();

        public void map(Object key, Text value, Context context

            ) throws IOException, InterruptedException {

            String line = value.toString();
```



```
// Split the line by delimiter and clean data

String[] fields = line.split(",");

String field1 = fields[0].trim();

String field2 = fields[1].trim();


// Write cleaned data to context

word.set(field1);

context.write(word, new Text(field2));

}

}

public static class DataCleaningReducer

    extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values,

        Context context

        ) throws IOException, InterruptedException {

        // Iterate over values and write to output

        for (Text value : values) {

            context.write(key, value);

        }

    }

}
```

```
}

public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "data cleaning");

    job.setJarByClass(DataCleaning.class);

    job.setMapperClass(DataCleaningMapper.class);

    job.setCombinerClass(DataCleaningReducer.class);

    job.setReducerClass(DataCleaningReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));

    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

DATASETS:

<https://www.kaggle.com/code/benhamner/competitions-with-largest-datasets>

REFERENCES:

1. <https://cloudxlab.com/blog/big-data-solution-apache-hadoop-and-spark/>
2. <https://cloudxlab.com/blog/introduction-to-big-data-and-distributed-computing/>
3. <https://github.com/PotatoSpudowski/The-centralized-guide-to-distributed-storage-and-processing-of-big-data>

Day 2:

Pyspark,

Dataset:

<https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset>

<https://www.kaggle.com/code/mrsohelrana/fashion-recommendation-system-using-bigdata-tech>

Day 3:

1. Explore generators in python
2. Explore chunk_size in pandas

GENERATORS:

Python generators can be used in distributed systems to efficiently process large amounts of data in a distributed and parallel manner. Generators in Python are a powerful feature that allow for lazy evaluation of data. This means that data is only generated as it is needed, rather than all at once. In a distributed system, this can be especially useful when dealing with large datasets that cannot fit into memory.

a generator only generates as many values as needed, which can be more memory-efficient than generating and storing all the values at once. For example, if you need to process a large file line by line, you could use a generator to read each line one at a time, rather than reading the entire file into memory at once.

Overall, generators can be a powerful tool in distributed systems for processing large amounts of data efficiently and in a parallel manner.

```
[2] !pip install pyspark
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.4.0.tar.gz (310.8 MB)
    310.8/310.8 MB 4.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.9/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.0-py2.py3-none-any.whl size=311317145 sha256=00c18095018b0154416a62141aef
  Stored in directory: /root/.cache/pip/wheels/9f/34/a4/159aa12d0a510d5ff7c8f0220abbea42e5d81ecf588c4fd884
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.0
```

```
[3] import pyspark
```

```
[4] from pyspark import SparkContext, SparkConf
```

```
def data_generator():
    # generate some data
    for i in range(100):
        yield i

conf = SparkConf().setAppName("example")
sc = SparkContext(conf=conf)

# distribute the generator across the cluster
data_rdd1 = sc.parallelize(data_generator())

# perform some processing on the data using Spark transformations and actions
result_rdd = data_rdd1.map(lambda x: x * 2).filter(lambda x: x % 3 == 0).collect()

# print the result
print(result_rdd)
```

```
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96, 102, 108, 114, 120, 126, 132, 138, 144, 150, 156, 162, 168]
```

PANDAS CHUNK SIZE:

When dealing with a large block of data that won't fit in the memory, you can divide the data into smaller portions using pandas' chunksize option.

Records? Or size?

Generators: size? Memory overhead?

<https://www.section.io/engineering-education/choosing-between-django-flask-and-fastapi/>
<https://docs.djangoproject.com/en/4.2/intro/tutorial01/>

<https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/>

REST :

REST stands for **representational state transfer** and is a software architecture style that defines a pattern for **client and server** communications over a network.

constraints:

- Stateless
- Client-server
- Cacheable
- Uniform interface
- Layered system: **proxy** or **load balancer**.

In REST, things are done using HTTP methods such as GET, POST, PUT, DELETE, OPTIONS, and, hopefully, PATCH.

REST pros

1. Decoupled client and server.
2. Cache-friendly.
3. Discoverability: no need of external documentation
4. Multiple formats support : for storing and exchanging data

REST cons:

1. No single REST structure.

2. Limited functionality (such as GET, POST, PUT, and DELETE)

3. Over- and under-fetching problems.

4. Security concerns

RESTful API and RESTless API are two different approaches to building web APIs.

RESTful API (Representational State Transfer) is a set of guidelines and architectural principles for building scalable and efficient web APIs. It emphasizes the use of HTTP methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources, and the use of hypermedia links to navigate between resources. RESTful APIs are designed to be stateless, meaning that the server does not store any information about the client between requests. RESTful APIs also support caching, which can improve performance by reducing the number of requests to the server.

On the other hand, RESTless API is a term sometimes used to describe APIs that deviate from the RESTful architectural principles. RESTless APIs may use different HTTP methods or rely on custom headers or query parameters to specify the desired operation. They may also rely on server-side sessions or cookies to maintain state between requests.

In summary, RESTful API is a well-established approach to building web APIs that follows a set of architectural principles, while RESTless API refers to APIs that deviate from these principles.

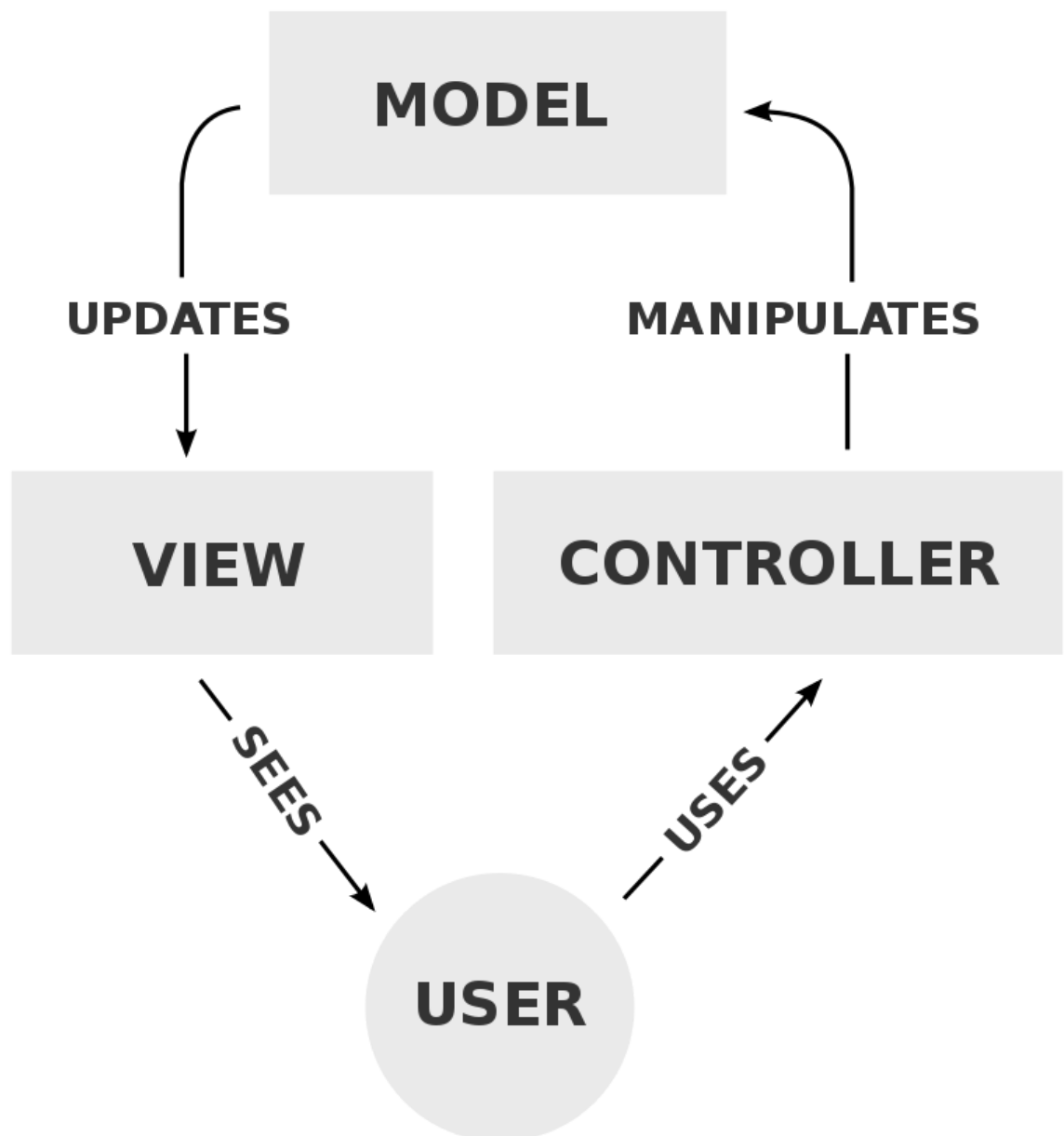
DJANGO:

Django is a web framework based on Python which is free and open source. The architecture pattern which it follows is the model-template-views architecture pattern.

easy to develop complex database-driven websites. The reasons for being famous are less code, low coupling and reusability, and pluggability of components in the time of development.

- Less code.

- The rapid development of websites.
- Pluggability of components in the time of development.
- Low coupling and reusability.



Pros

- 1.automatically enable the security firewall
- 2.It equipped the rest framework with influential [FastAPI](#) functionality

Cons:

- 1.may limit development speed. because of the many reusable modules.
- 2.Django Models can have no mixins, they just have simple inheritance.
- 3.Django is not recommended for small works and websites.due to its complexity
- 4.minor mistake and mismatch can lead to the failure of website
- 5.Django offers numerous features and specifications, which is why it is very hard to understand for a user.

Django: Use cases

Django can tackle projects of any size and capacity. We can use it for simple sites or high-performance sites. A few example use cases would be:

[High load booking engines.](#)

[Shopping platforms.](#)

[School management systems.](#)

Built-in custom CRM systems for internal data.

IOS and Android applications supporting web-based applications.

Step 1:

```
django-admin startproject myproject
```

Step 2:


```
cd myproject
```

```
python manage.py startapp myapp
```

Step 3:

```
from django.http import HttpResponse
```

```
def hello(request):
```

```
    return HttpResponse("Hello, World!")
```

Step 4:

```
from django.urls import path
```

```
from myapp.views import hello
```

```
urlpatterns = [
```

```
    path('hello/', hello, name='hello'),
```

```
]
```

Step5 :

```
python manage.py runserver
```

Flask

a micro framework used to develop small sizes or light applications. All the functionalities can be done using pre-existing third-party libraries. Flask comes with options like template engines such as ORM, caching, and authentications.

Pros

- 1.Flask is flexible and comfortable. Most parts of Flask have the possibility of changing, which is very unlikely for some other web frameworks.
- 2.Flask allows unit testing
- 3.Flask is very beginner-friendly because of its simplicity, enables developers to create apps effortlessly and quickly.

CONS:

- 1.Flask uses Modules, which is a third-party involvement and is prone to cause security breaches.
- 2.Flask has a singular source that implies that it will handle every request in turns, one after the other, so regardless of how many multiple requests, it still takes them in turns, which takes more time.
- 3.It has a few tools, due to which a developer needs to look for tools in other libraries

Flask: Use cases

We can use Flask for commercial projects. It can help you get started quickly but doesn't work well when resembling a real load.

You can quickly implement Flask projects such as:

- E-commerce systems.
- Facebook/Twitter bot.
- An online social network.
- Static sites.

It's not advisable to use it for high-load enterprise software.

FastAPI

Fast API is a modern, open-source, fast, and highly performant Python web framework used for building Web APIs and is based on Python 3.6+ standard type hints.

- Fast development
- High and fast performance
- Fewer bugs

Pros:

Fast API validates the developer's data type even in deeply nested JSON requests.

Fast API is built on standards like JSON Schema (a tool used for validating the structure of JSON data), OAuth 2.0 (it's the industry-standard protocol for authorization), and OpenAPI (which is a publicly available application programming interface)

JSON Schema: It allows a developer to annotate and substantiate JSON documents. JSON Schema is like a vocabulary.

OpenAPI: Open API is an interface available for the public for programming that provides access to modify the application according to the requirements.

It offers an autocomplete specification that helps the developers

FastAPI makes it easy to build a GraphQL API with a Python library called *graphene-python*.

Cons:

Because FastAPI is relatively new, the community is small compared to other frameworks, and regardless of its detailed documentation, there is very little external educational materials.

comparison:

Performance:FastAPI

Community: Django

Flexibility:Flask(modifications in future)

Packages: Django has the most packages in its library, which count around 2500.

.